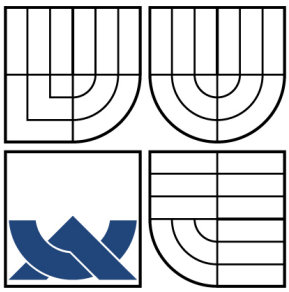
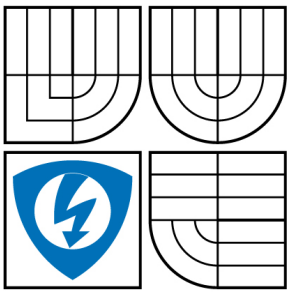


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

SPECIFIKA A ROZDÍLY VÝVOJE SERVEROVÝCH APLIKACÍ NA PLATFORMĚ JAVASE A .NET

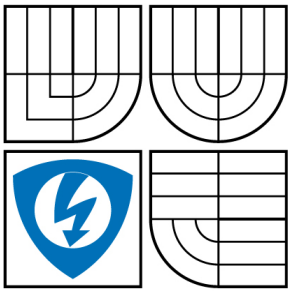
SPECIFICS OF AND DIFFERENCES IN THE DEVELOPMENT OF SERVER APPLICATIONS ON THE JAVASE AND .NET PLATFORMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ONDREJ VARGA
AUTHOR

VEDOUCÍ PRÁCE — Ing. PETR KOVÁŘ
SUPERVISOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student:	Varga Ondrej	***ID:***	83162
Ročník:	3	***Akademický rok:***	2007/2008

NÁZEV TÉMATU:

Specifika a rozdíly vývoje serverových aplikací na platformě JavaSE a .NET

POKYNY PRO VYPRACOVÁNÍ:

Prozkoumejte možnosti vývoje serverové strany síťových aplikací na platformách JavaSE a .NET. Vytvořte na obou platformách aplikaci, která zajistí základní služby FTP serveru a získané poznatky srovnejte se závěry z teoretické části práce.

DOPORUČENÁ LITERATURA:

[1] HEROUT, P.: Učebnice jazyka JAVA, Kopp, České Budějovice 2000, 352s. ISBN 80-7232-115-3.
[2] HAROLD, E.R.: Java Network Programming, O'Reilly, 2004, 760s., ISBN 0-596-00721-3.
[3] REID, F.: Network programming in .NET: C# & Visual Basic .NET, Digital Press, 2004, 541s., ISBN 1555583156.
[4] POSTEL, J., REYNOLDS, J., RFC 959 - File Transfer Protocol [online], The Internet Engineering Task Force, 1985. <URL: http://www.ietf.org/rfc/rfc0959.txt?number=0959>

Termín zadání: ***Termín odevzdání:***

Vedoucí práce: Ing. Petr Kovář

prof. Ing. Kamil Vrba, CSc.
předseda oborové rady

UPOZORNĚNÍ:

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

1. Pan/paní

Jméno a příjmení: Ondrej Varga
Bytem: Péczeliho 4, 94501, Ďulov Dvor
Narozen/a (datum a místo): 2.12.1985, Komárno

(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta elektrotechniky a komunikačních technologií
se sídlem Údolní 244/53, 60200 Brno 2
jejímž jménem jedná na základě písemného pověření děkanem fakulty:
.....

(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   - ☐ disertační práce
   - ☐ diplomová práce
   - ☑ bakalářská práce

   jiná práce, jejíž druh je specifikován jako .....

(dále jen VŠKP nebo dílo)

Název VŠKP: Specifika a rozdíly vývoje serverových aplikací na platformě JavaSE a .NET
Vedoucí/školitel VŠKP: Ing. Petr Kovář
Ústav: Ústav telekomunikací
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:

- ☐ tištěné formě - počet exemplářů .....
- ☐ elektronické formě - počet exemplářů .....

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.

4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti
   - ☑ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Anotácia

V dnešnej dobe modernej technológie by sme asi ťažko našli človeka, ktorý by si vedel predstaviť svoj život bez počítača. Počítačové aplikácie sú vytvorené pomocou neustále sa vyvíjajúcich programovacích jazykov. Medzi najnovšími a najmodernejšími platformami patria JavaSE a .NET, ktoré použijem podľa úlohy aj pri spracovaní tejto bakalárskej práci.

Najnovšie platformy sú použiteľné vo väčšine oblastí počítačových programov, aj v sieťových aplikáciách. Sú v nich implementované triedy zaoberajúce sa práve touto oblasťou. V týchto triedach sú už niektoré príkazy dopredu naprogramované, a tým je jednoduchšia naša práca pri programovaní takýchto aplikácií.

V dnešnej dobe veľmi často používanou sieťovou aplikáciou je FTP server, ktorý používa FTP (File Transfer Protocol) pre komunikáciu s klientom. Je dobre stanovený internetový protokol určený na prenášanie dáta (a informácie o súboroch) cez počítačové siete, ktoré používajú TCP.

Cieľom nasledujúcej bakalárskej práce je preskúmať možnosti vývoja serverovej strany sieťových aplikácii na platformách JavaSE a .NET. To znamená, naprogramovať FTP server zaisťujúci základné služby na obidvoch platformách, a potom zhrnúť výhody a nevýhody jednotlivých platforiem pri programovaní tejto sieťovej aplikácie.

Kapitola č.1 popisuje teoreticky platformu JavaSE, programovací jazyk Java a použité vývojové prostredie NetBeans. Vlastnosti platformy .NET, obecná charakteristika jazyka C# a vývojové prostredie Visual Studio sú popísané v kapitole č.2. Kapitola č.3 obsahuje realizáciu praktickej časti úlohy. Ďalšia kapitola sa zaoberá teóriou sieťovania a obecnou charakteristikou protokolu FTP. V poslednej kapitole sú zhrnuté dosiahnuté výsledky a vedomosti.

Kľúčové slová: JavaSE, .NET, FTP, server, klient

Abstract

Nowadays, in this modern world full of technology, it would be very difficult to find someone, who could live without using the computer. The computer applications are made by the always developing programming languages. The one of the newest and the most modern platforms are the JavaSE and the .NET, which two I needed to use for this bachelor's thesis.

The newest developing platforms can be used in a lot of section of computer programs, in the network applications as well. Some classes dealing with exactly this section are implemented in them. Some of the commands are already programmed in these classes, so it helps to be easier our job at programming an application like this.

Nowadays, a very frequently used network application is the FTP server, which uses FTP (File Transfer Protocol) for the communication with the client. It is a well established internet protocol defined for transfer data (and information about files) thorough computer networks, which uses TCP.

Goal of this bachelor's thesis is to check abilities of development of the server side of the network applications on the JavaSE and .NET platforms. It means, to develop an FTP server application on both platforms, which includes the basic services, and after programming this network application resume the pros and cons of the used platforms.

Chapter 1 describes the theory of platform JavaSE, developing language Java and used developing environment NetBeans. Properties of platform .NET, developing language C# and developing environment Visual Studio are described in chapter 2. Chapter 3 contests the realization of the practical part of the layout. Next chapter includes the basics of networking and the general characteristic of the FTP. In the last chapter are all the achievements and knowledge.

Key words: JavaSE, .NET, FTP, server, client

BIBLIOGRAFICKÁ CITACE PRÁCE

VARGA, O. *Specifika a rozdíly vývoje serverových aplikací na platformě JavaSE a .NET*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. 49 s. Vedoucí bakalářské práce Ing. Petr Kovář.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Specifika a rozdíly vývoje serverových aplikací na platformě JavaSE a .NET“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.“

V Brně dne …… ……..

(podpis autora)

ZOZNAM POUŽITÝCH SKRATIEK

API	Application Programming Interface
ASCII	American Standard Code for Information
BCL	Basic Class Library
CLR	Common Language Runtime
COM	Component Object Model
DTP	Data Transfer Process
EBCDIC	Extended Binary Coded Interchange Code
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IP	Internet Protocol
JavaSE	Java Standard Edition
JDK	Java Development Kit
JIT	Just in Time
JRE	Java Runtime Environment
JVM	Java Virtual Machine
MSIL	Microsoft Independent Language
NC	Net computer
PC	Personal computer
PI	Protocol Interpreter
RFO	Request For Comments
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UML	Unified Modeling Language
URL	Uniform Resource Locator
WWW	World Wide Web

OBSAH

ZOZNAM OBRÁZKOV

Úvod

Ľudstvo žije v takej dobe, kde programovanie má stále väčší význam, a to nielen v informatike ale aj v elektrotechnike. Napríklad pomocou programov môžeme vypočítať rýchlo a jednoducho aj veľmi dlhé a komplikované výpočty. Takže veľkou prednosťou elektrotechniky je, keď pozná nejaký programovací jazyk.

V mojom projekte mám za úlohu používať programovanie na iný účel ako v predchádzajúcom príklade. Cieľom úlohy bolo preskúmať možnosti vývoja serverovej strany sieťových aplikácii na platformách JavaSE a .NET. Mal som vytvoriť na oboch platformách aplikáciu, ktorá zaistí základné služby FTP servera.

Projekt je rozdelený na dve tematiky. V prvej som sa podrobne venoval platforme JavaSE a jeho možnostiam sieťového programovania. V druhej časti sa venujem tomu istému problému, ale na platforme .NET.

Obe platformy obsahujú objektovo orientované a relatívne nové programovacie jazyky (Java a C#), takže medzi vývojom dvoch FTP serverov by nemal byť veľký rozdiel. Ten pravdepodobne bude hlavne v syntaxe a možno v niektorých implementovaných triedach a metód.

Za účelom získania hlbšej analýzy spomínaných rozdielov, som naprogramoval FTP server na obidvoch platformách, a pri ich vyvíjaní som skúšal zistiť odlišnosti ich sieťového programovania.

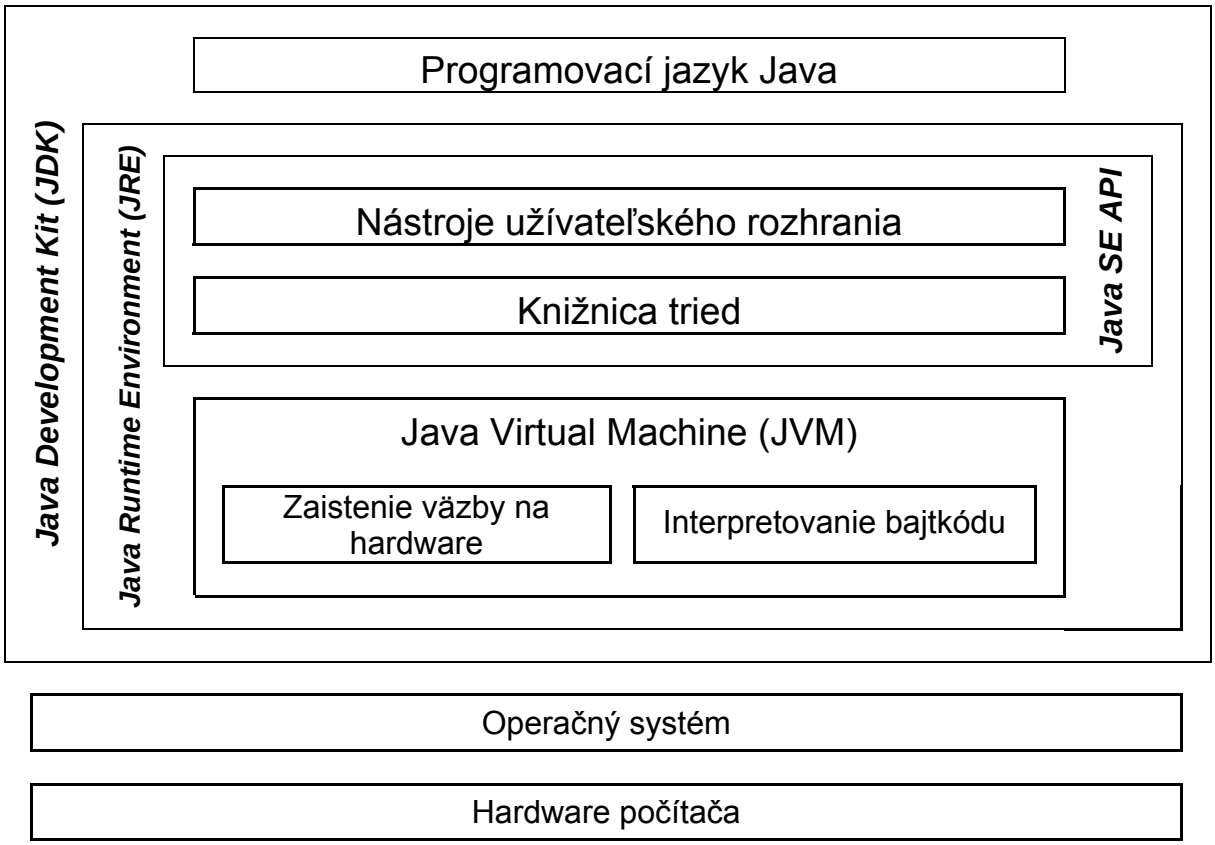
1 Platforma JavaSE

1.1 Štruktúra platformy

Podmienkou pre programovanie v jazyku Java a pre spúšťanie v ňom vytvorených programov je inštalácia prostredia JavaSE, tak isto ako v jazyku C# prostredie .NET Framework.

Jedným z najviac oceňovaných prínosov Javy je plná prenositeľnosť programov na ľubovoľnú platformu bez potreby ich prekladu na tejto platforme. Prenositeľnosť je riešená tak, že preklad neprebieha do strojového jazyka počítača , ale do pseudojazyka nazývaného *byte-code* (*bajtkód*), ktorý je uložený v súboroch s vyhradenou príponou *.class*. Tento súbor je potom z disku zavedený do pamäte počítača a súčasne prebieha overenie bajtkódu na platforme. Jeho interpretácia je potom úlohou špeciálnych programov, ktoré sa súhrne nazývajú *Java platforma*.

Táto platforma sa skladá z dvoch hlavných častí: Java Runtime Environment (JRE) a Java Development Kit (JDK):



Obr. 1: Štruktúra platformy JavaSE

Java SE Runtime Environment (JRE)

Java SE Runtime Environment poskytuje základnú knižnicu tried, Java Virtual Machine a ďalšie komponenty, ktoré sú potrebné pre beh aplikácií písané v programovacom jazyku Java.

Dôležitou súčasťou JRE je knižnica tried, čo zahŕňa ***Java SE API.*** Táto skratka znamená *Application Programming Interface* (*aplikačné programové rozhranie*). Táto časť obsahuje veľké množstvo tried, ktoré sú považované za štandardné, takže sa musí vyskytovať v každom prostredí, kde sa Java používa.

Keď náš program používa metódy z API, tak nie je ich kód súčasťou nášmu programu, pretože je súčasťou API. Prakticky to znamená, že naše programy obsahujú len kód, ktorý sme napísali my, a súbory, v ktorých sú naše preložené programy uložené, preto majú pomerne malú veľkosť.

Úlohou prostredia ***Java Virtual Machine***, tzv. *virtuálny stroj*, je zaistenie hardware a operačný systém nezávislosť platformu Java SE, malú veľkosť prekladaných kódov a zabezpečenie. JVM je abstraktným počítacím strojom, ktorý obsahuje sadu príkazov

a manipuluje pamäť pri spustení programu. Skladá sa z časti zaisťujúce väzbu na hardware a z časti interpretujúci bajtkód. Tento interpreter môže byť nahradený JIT kompilátorom.

Problémom interpretovaných jazykov je ich pomalosť v porovnaní s kompilovanými jazykmi. Java tento problém čiastočne rieši použitím tzv. *JIT kompilátorov* (*Just In Time*), ktoré v dobe zavádzania programu z disku do pamäte počítača (po overení správnosti bajtkódu) ju preloží do strojového jazyka konkrétneho počítača, čím z nej prakticky vyrobí v pamäti .exe program. Ten potom beží podobnou rýchlosťou, ako ktorýkoľvek iný kompilovaný program napísaný trebárs v jazyku C.

Java Development Kit (JDK)

JDK je nadriadeným modulom JRE, ktorý obsahuje všetko čo je v JRE a ešte naviac aj nástroje potrebné pre vývoje aplikácií.

1.2 Programovací jazyk Java

1.2.1 História jazyka

Základy Javy je možné nájsť v projekte Oak, ktorý vznikol vo firme Sun na začiatku deväťdesiatych rokov pre riadenie elektronických výrobkov. V roku 1994 bol prenesený ako programovací jazyk do prostredia počítačov pod názvom Java (horká káva).

Veľmi významným faktorom pre rozvoj používania Javy sa stalo v roku 1995 zaradenie jej podpory do vtedy veľmi populárneho prehliadača Netscape Navigátor 2.0. Táto podpora umožňovala rozšírenie funkčnosti webových stránok, pomocou java appletov, programov v Jave, sťahovaných súčasne s WWW stránkou a spustených priamo v prehliadači na strane klienta. Neskoršie táto podpora bola už zavedená aj do ďalších prehliadačov a applety sa stali nedeliteľnou súčasťou internetových strán.

Tento mechanizmus dokonca viedol k vzniku myšlienky NC (net computers), ktorý mal existovať bez pevných diskov, operačného systému a lokálne inštalovaných programov. Ďalej mal obsahovať len integrovaný internetový prehliadač a všetky programy sa mali cez neho spúšťať zo sieťových serverov vo forme java appletov. Ako výhody sa okrem úspor na hardware komponentoch uvádzala zjednodušená administrácia počítačových sieti, zvýšená bezpečnosť, rýchla inštalácia a aktualizácia programov ako aj ďalšie dôvody. Táto vízia sa neujala, ceny týchto NC bez diskovej mechaniky neboli o mnoho nižšie ako štandardné PC, a

čo bolo podstatnejšie, že nedošlo k masovému prepísaniu aplikačného software do formy appletov.

Aj cez neúspech tohto pokusu (ktorý by mal šancu nahradiť stávajúcu platformu PC) si Java svoje miesto udržala, na WWW stránkach našla svoju "parketu" v plne internetových aplikáciách, ktoré sprostredkovávajú komunikáciu medzi klientom na internetovom prehliadači a službách prístupných cez internetový server. Tu sa využili výhody Javy, jej robustnosť, stabilita, rozsah funkcií a hlavne bezpečnosť. Aj preto podporu Java appletov nájdeme v najväčšej miere na WWW stránkach internetových bankovníctvach a v ďalších aplikáciách vyžadujúcich vysokú mieru stability a zabezpečenia.

Java sa však neobmedzuje len na java applety, práve naopak. Jej výhodu, multiplatformitu, deklarovalo populárne heslo : Write once run everywhere (mohli by sme preložiť ako „raz to napíšeš a spustíš ho všade"). Zdrojový kód je pri vývoji preložený do spustiteľného medzikódu (bytecode), ktorý je možné potom spúšťať pomocou nainštalovaného runtime prostredia (Java Virtual Machine), priamo na rôznych typoch počítačov či technických zariadeniach bez nutnosti nového prekladu.

1.2.2 Vlastnosti jazyka

Java je vyspelý programovací jazyk, obsahujúci všetky vlastnosti, ktoré sú vyžadované v modernom programovaní, od modularity programu, riadiacich konštrukciu, cez silné typové kontroly, multithreading, ošetrenie výnimiek, správu pamäti, i silnú podporu pre databázy a sieťové operácie.

K jej výhodám patrí okrem už spomínanej multiplatformity jej robustnosť, škálovateľnosť a vysoká bezpečnosť, ktorá ju profituje pre používanie na kritické aplikácie kritických aplikácií na mainfraimových počítačoch.

Nižšia rýchlosť, spôsobená spracovaním v runtime prostredia môže byť urýchlená pomocou špecializovaných prekladačov na cieľovom prostredí (Java just-in-time). Aj keď základné vývojové prostredie obsahuje len riadkový prekladač, existuje mnoho vývojových nástrojov a rozšírenie ďalších firiem autorov .

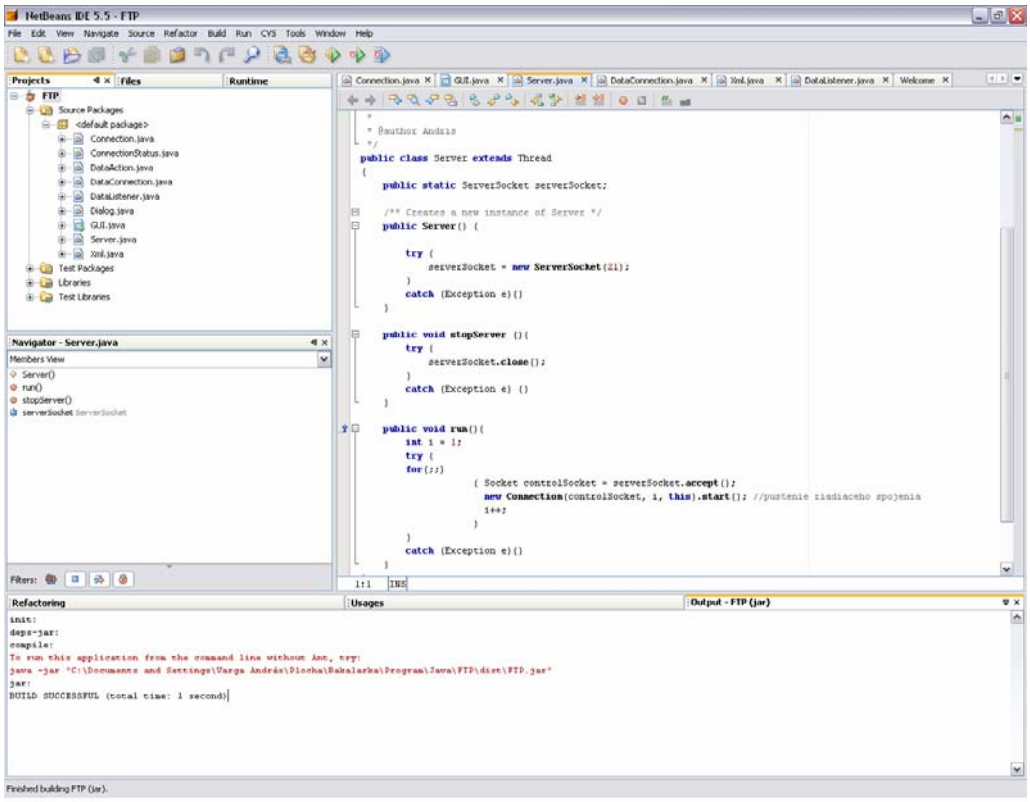
Java je predovšetkým silne objektová, čo umožňuje v nej modelovať, vytvárať, používať a rozširovať rozsiahle knižnice a systémy. Práve objektovo je potrebné myslieť nie len pri písaní programu, ale už pri návrhu a analýze. Pre tieto účely boli vytvorené UML,

Unified Modeling Language, modelovací jazyk slúži k objektovému modelovaniu a popisu konštrukcií reálneho sveta, prevádzaných do sveta počítačov a informačných systémov.

1.3 Vývojové prostredie NetBeans

Moja aplikácia bola vytvorená v prostrediu Netbeans IDE 5.5. Ide o voľne šíriteľný nástroj (licencia Open source), ktorý je zdarma stiahnuteľný zo stránkach výrobcu: http://www.netbeans.org.

Prostredie sa špecializuje na programovací jazyk Java, ktorý umožňuje tvorbu najrôznejšieho softwaru od mobilných až po serverové aplikácie. Stojí za ním silná komunita vývojárov, ktorá ju neustále zdokonaľuje a rozširuje jeho možnosti. Obsahuje radu rôznych užitočných nástrojov a doplnkov, čím poskytuje programátorom veľmi príjemné a pohodlné programovanie.

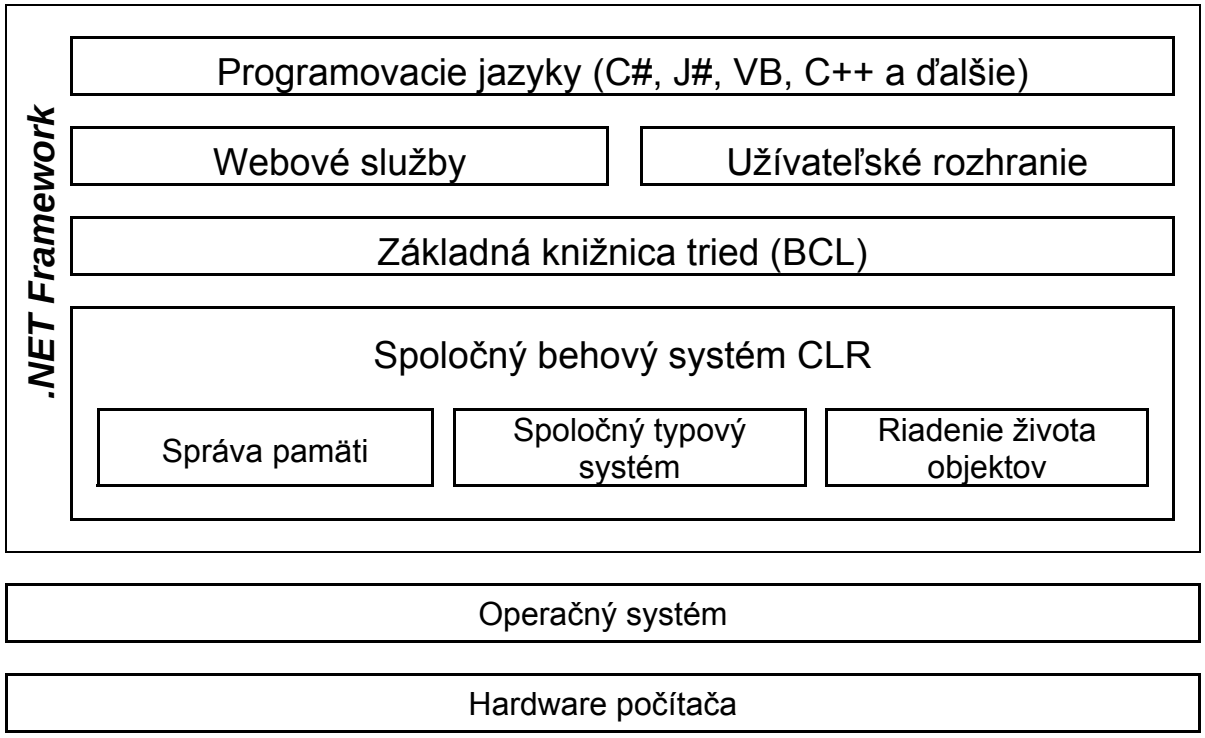


Obr. 2: Vývojové prostredie NetBeans

2 Platforma .NET

2.1 Štruktúra platformy

Dôležitou podmienkou pre programovanie v jazyku C# a pre spúšťanie vytvorených aplikácii v tomto jazyku je inštalácia prostrediu .NET Framework. Toto prostredie sa skladá z niekoľko súčastí:



Obr. 3: Štruktúra prostredia . NET Framework

Jadrom prostredia .NET Framework je spoločné behové prostredie, po anglicky označované *Common Language Runtime* (CLR). Toto prostredie zaisťuje beh programov preložených z rôznych programovacích jazykov do univerzálneho pomocného jazyka označovaného Microsoft Intermediate Language (MSIL alebo len IL). CLR umožňuje ich vzájomnú spoluprácu, takže rôzne súčasti programu môžu byť napísaný v rôznych programovacích jazykoch. Medzijazyk sa stará o automatickú správu pamäti, o riadenie doby života objektov a o ďalšie veci potrebné pre beh programov v tomto prostredia.

Prenositeľnosť tejto platformy je riešená tak, že zdrojový kód napísaného v jednom z uvedených jazykoch v tabuľke sa preloží, ale nie do strojového kódu počítača, ale do

medzijazyka MSIL. Ten sa pomocou ďalšieho prekladača prevedie do strojového kódu cieľového počítača - ale až v tom momente, keď program spustíme.

Kód, vzniknutý prekladom programu napísaného v C# sa nazýva *riadený kód* (*managed code*). Tento kód beží pod dozorom CLR a využíva jeho služby.

Ďalšou dôležitou súčasťou prostredia .NET Framework je knižnica tried, nazývaná súhrnne *Základná knižnica tried* (*Basic Class Library*). Tieto triedy pokrývajú všetko od vstupných a výstupných operácií po kontajnery (triedy slúžiace pre ukladanie rôznych dát). Vzhľadom k tomu jazyk C# prakticky nepotrebuje svoju vlastnú knižnicu.

Nad knižnicou BCL sú ešte knižnice pre tvorbu grafického užívateľského rozhrania programov a pre webové služby.

Najvyššiu vrstvu prostredia .NET Framework tvoria prekladače z rôznych programovacích jazykov a prekladače JIT.

Jazyk C# - a prostredie .NET Framework – umožňuje spolupracovať i s tzv. *neriadeným kódom* (*unmanaged code*). To je kód, ktorý nebeží v rámci CLR; môže ísť o staršie programy.

Súčasťou prostredia .NET Framework sú tiež mechanizmy, ktoré sa starajú o zabezpečenia distribuovaných aplikácii.

2.2 Programovací jazyk C#

2.2.1 História jazyka

Jazyk C# je novým programovacím jazykom firmy Microsoft. Prikladá sa mu veľká dôležitosť a budúcnosť – sama firma Microsoft tvrdí, že sa stane spolu s C++ ich interným programovacím jazykom. Jazyk C# bol vytvorený s úmyslom prevziať dobré vlastnosti všetkých existujúcich jazykov.

Najviac je C# podobný jazyku Java. Vznikol na základe podobných myšlienkových úvah a snahy riešiť podobné problémy. Jeho syntax vychádza z jazyka C++, zmeny hlavne smerujú k zjednodušeniu jazyka. Ďalšie odlišnosti boli evokované snahou o jednoduchú podporu komponentovo orientovaného vývoja – tu samozrejme ide o technológiu COM firmy Microsoft a jazyk C# sa inšpiroval v jazyku Visual Basic.

2.2.2 Vlastnosti jazyka

Jazyk C# veľmi dobre zapadá do stratégie .NET firmy Microsoft. Programy napísané v jazyku C# pre svoj beh vyžaduje .NET Framework. Ide o platformu, nad ktorou sa spúšťajú programy, v Jave je to Java Runtime.

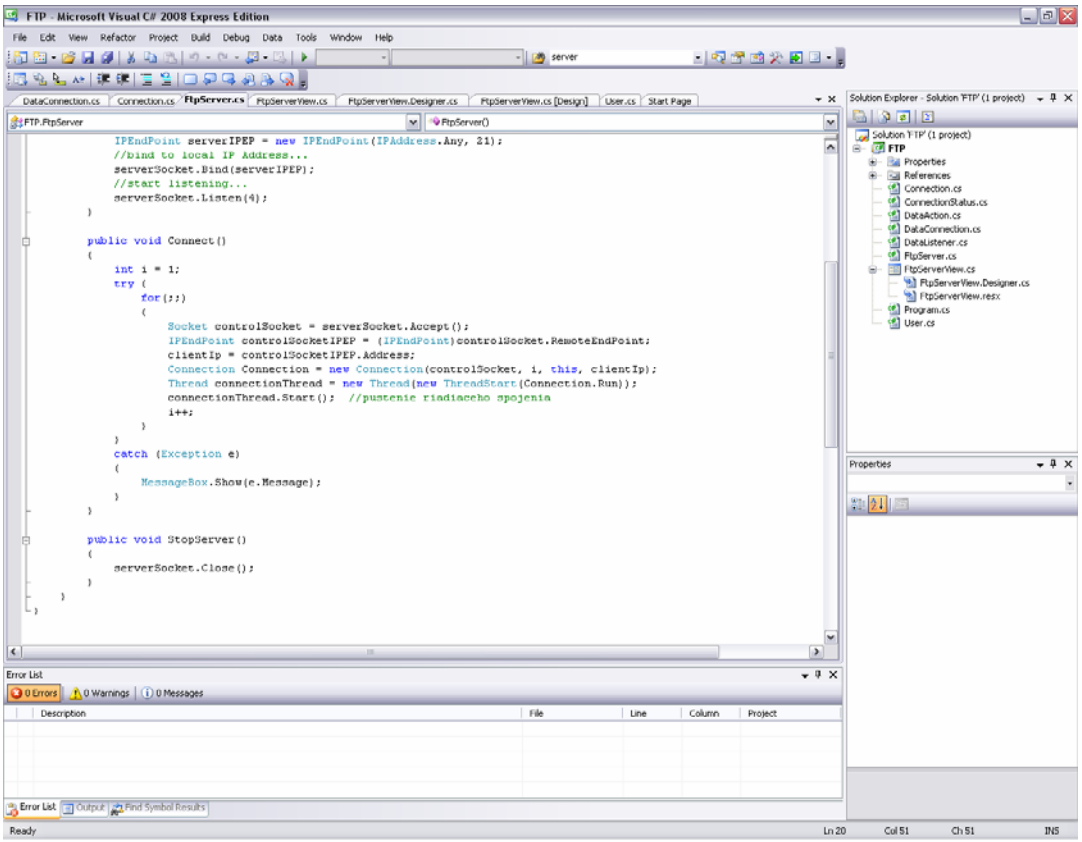
Na odlišnosť od Java Runtime existuje viac jazykov, ktoré je možné použiť pre vytváranie programov pre túto platformu. Vďaka spoločnému typovému systému a rovnakému binárnemu rozhraniu je jazyková nezávislosť doťažená až do takej podoby, že je možné bez problémov jeden projekt vytvárať v rôznych programovacích jazykoch.

Proces spustenia aplikácie je tiež sofistikovanejší. Nielenže je možné určiť, či sa bude medzikód (tu MSIL = Microsoft Independent Language) interpretovať, alebo pred spustením prekladať pomocou JIT kompilátora. Existuje dokonca možnosť kompilovať len časť kódu, ktorá je aktuálne potrebná a za chodu programu doplňovať preklady časti kódu, ktoré ešte neboli preložené.

Jazyk C# je veľmi podobný jazyku Java. Jeho orientácia je trošku odlišná. Kým Java sa pred pár rokmi prezentovala ako zaujímavá alternatíva k vytváraniu bežných aplikácií, a napriek tomu, že sa v mnohých oblastiach veľmi osvedčila, nikdy sa nestala nástrojom väčšiny programátorov. Jazyk C# oproti tomu mieri do mainstreamu. Je možné s ním vytvárať ako „obyčajné“ aplikácie, tak aj webové služby, alebo aktívne webové stránky.

2.3 Vývojové prostredie Visual Studio 2008

Pri práci na platforme .NET som používal Visual C# 2008 Express Edition, ktorý je vývojový nástroj základnej úrovne. Je zdarma k dispozícii a stiahnuteľný zo stránok Microsoftu: http://www.microsoft.com/express/download. Je určený začiatočníkom a hodí sa k vytváraniu jednoduchých aplikácií pre Windows a jednoduchých interaktívnych webových stránok. Nástroj nie je svojim charakterom ani svojimi vlastnosťami určený pre vývoj zložitejších aplikácií, aj keď ich tvorba v Express nástrojoch, ani ich následná distribúcia nie je licenčne obmedzená. Toto prostredie je určené práve pre študentov, takže som si ho vybral pre vývoj môjho servera.



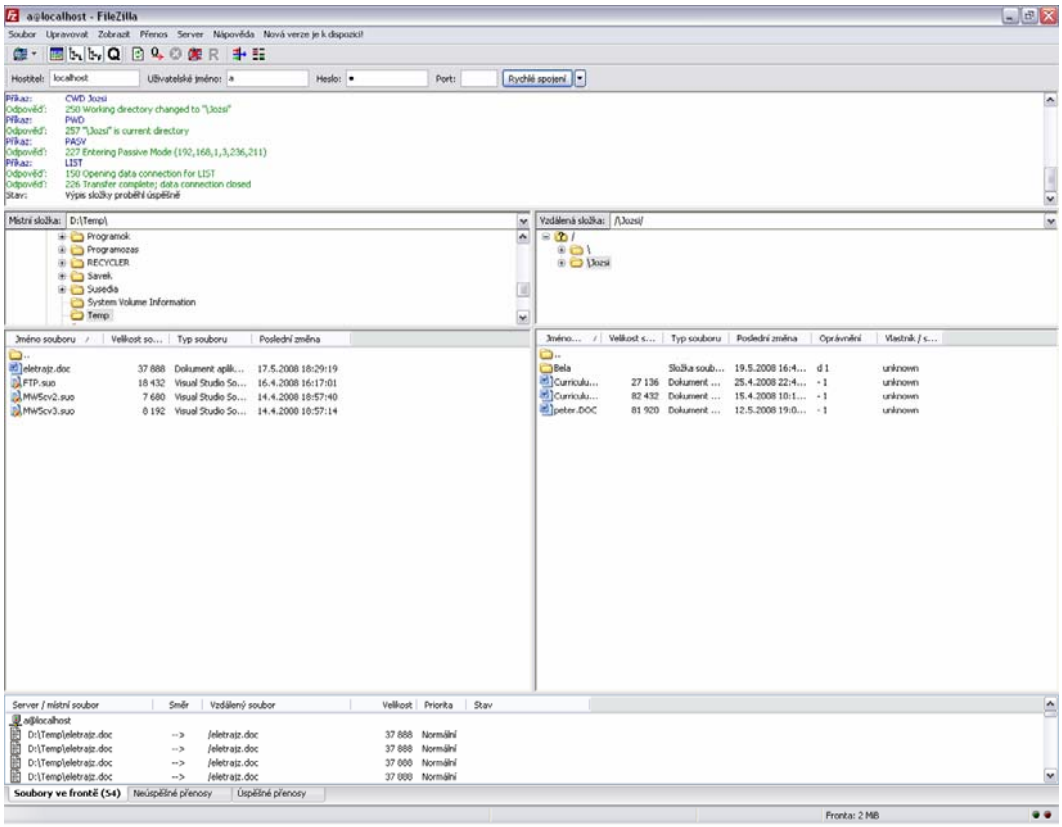
Obr. 4: Vývojové prostredie Microsoft Visual Studio 2008

3 Naprogramované FTP servery

3.1 Používaný FTP klient

K vývoju serverov som používal freeware aplikáciu FileZilla Client 3.0.8.1, ktorá je zdarma stiahnuteľná zo stránky výrobcu: http://filezilla-project.org .

Aplikácia je ľahko ovládateľná a obsahuje všetky základné funkcie, ktoré boli potrebné na testovanie pri vývoji mojich programov. Tento klient predvolene pracuje v pasívnom režime, ale v jeho nastaveniach sa dá nastaviť aj aktívny režim.



Obr. 5: Aplikácia FileZilla

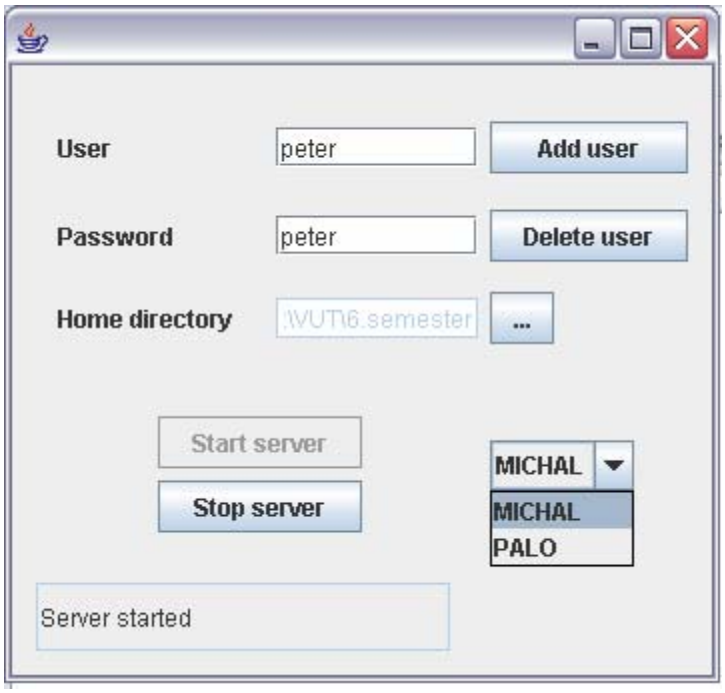
3.2 FTP server na platforme JavaSE

3.2.1 Užívateľské rozhranie

Po kompilácii a spustenia zdrojového kódu sa automaticky spustí užívateľské rozhranie servera, kde sú rozmiestnené v Jave dopredu definované komponenty: jButton, jLabel, jTextField a jComboBox. Toto rozhranie sa skladá z dvoch častí:

1. Práca s databázou užívateľov

2. Spustenie a zastavenie servera



Obr. 6: Java FTP server

3.2.2 Práca s databázou užívateľov

Server pracuje s xml databázou užívateľov, ktorá je uložená do jedného xml súboru. Tento súbor sa nazýva userlist.xml a má byť uložený v adresáru *C:\Temp*. V zdrojovom kóde všetky metódy ohľadom práci s databázou zahrňuje trieda Xml. Keď súbor userlist.xml je

umiestnený na správnom mieste, tak ho program načíta pomocou metódy loadXml( ), a potom s ním pracuje ako svojou databázou.

Aby sa užívateľ mohol pripojiť k serveru, musí mať svoje užívateľské meno, heslo a domovský adresár v databáze. Pomocou tlačítka *Add user* a *Delete user* môžeme pridávať alebo odobrať užívateľov, ktorých chceme, aby mali prístup k serveru. Stalčením týchto tlačítok sa zavolajú metódy addUser( ), resp. deleteUser( ), ktoré upravujú obsah xml súboru.

Zoznam užívateľov je vypísaný do tzv. *jComboBoxu*, ktorého obsah je aktualizovaný pomocou metódy getUserlist( ). Aktualizácia je prevedená hneď pri spustení užívateľského rozhrania a vždy po stlačení tlačítok *Add user* a *Delete user*.

3.2.3 Spustenie a zastavenie servera

Na spustenie servera slúži tlačítko *Start server*:

```
server = new Server( );
server.start( );
```

Po stlačení tlačítka sa vytvorí nový konštruktor typu Server a pustí sa jeho vlákno.

```
serverSocket = new ServerSocket(21);
```

Konštruktor typu Server obsahuje jediný príkaz, ktorý vytvorí nový serverSocket na porte 21 (port vyhradený pre FTP). Tento socket sa automaticky začne počúvať na danom porte a čakať na pripojenia od klientov.

```
Socket controlSocket = serverSocket.accept( );
new Connection ( controlSocket, i, this ).start( );
```

V serverovom vlákne je implementovaná metóda na vytvorenie riadiaceho spojenia s klientom. Serverov socket a čaká dovtedy, kým sa k nemu pripojí jeden klient, potom ho akceptuje a vytvorí sa konštruktor triedy Connection, ktorý reprezentuje riadiace spojenie. Zároveň sa pustí vlákno riadiaceho spojenia, kde sa prenášajú všetky príkazy FTP pri komunikácii s klientom.

Na zastavenie servera slúži tlačítko *Stop server*:

```
serverSocket.close( );
```

Zavolaná je metóda close( ), ktorá zatvorí serverov socket, a tým pádom je server zastavený.

3.2.4 Pripojovanie klienta (riadiace spojenie)

Keď je server už spustený a pripravený na prijímanie spojení, tak sa užívateľ môže pripojovať k danej ip adresy a k portu so svojím prihlasovacím menom a heslom. Server vo svojom vlákne najprv akceptuje klientov socket, a potom vytvorí nové riadiace spojenie s klientom, čím sa spustí ďalšie vlákno pre toto spojenie s aktuálnym klientom.

```
BufferedReader controlIn = new BufferedReader ( new InputStreamReader (
        controlSocket.getInputStream( ) ) );
controlOut = new PrintWriter ( controlSocket.getOutputStream( ), true );
```

Komunikácia cez riadiace spojenie je vyriešené pomocou streamov. Na prijímanie sa používa premenná typu InputStreamReader, ktorá hneď ukladá prijaté streamy do BufferedReaderu. Na odoslanie streamov ku klientovi slúži contolOut, typu PrintWriter. Cez toto spojenie prechádzajú všetky príkazy FTP, ktoré sú potrebné pre pripojovanie klienta a pre rôzne iné služby okrem dátových.

3.2.5 Prenášanie dát (dátové spojenie)

(FilaZilla klient štandardne pracuje v pasívnom režime, takže predpokladajme, že ide o pasívny režim – aktívny režim by bol podobný.)

Server po prijatí príkazu PASV sa pripraví na pasívny režim:

```
dataListener = new DataListener(serverIp);
dataListener.start( );
```

Vytvorí nový konštruktor triedy **DataListener** so serverovou ip adresou, v ktorom je vytvorený nový serverSocket pre prípadný dátový prenos. Pustí sa vlákno tohto konštruktoru, v ktorom server čaká na danom ip adrese a porte na prípadný prenos.

```
dataConnection = new DataConnection ( this, dataListener.getSocket( ),
        f, DataAction.RETR );
```

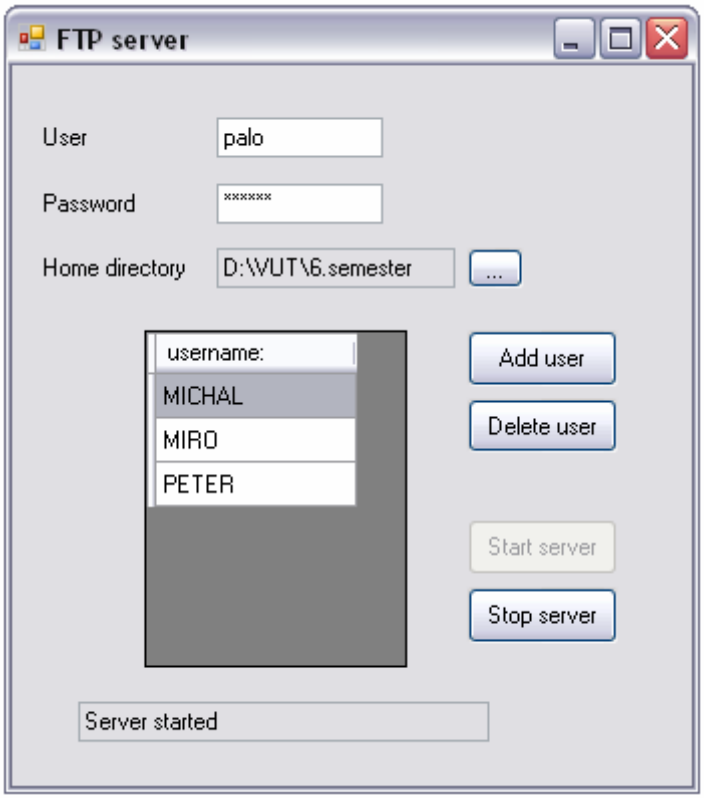
dataConnection.**start( )**;

Prenášanie dát je realizované vo vlákne konštruktoru triedy **DataConnection**, kde sa uskutočnia oba typy prenosu (retrieve a store).

3.3 FTP server na platforme .NET

3.3.1 Užívateľské rozhranie

K vyhotoveniu grafického užívateľského rozhrania som používal komponenty z Toolboxu vývojového prostredia Visual Studio: Button, TextBox, Label a DataGridView. Tieto komponenty sú podobne rozmiestnené a majú podobné funkcie ako v predošlom programu.



Obr. 7: C# FTP server

3.3.2 Práca s databázou užívateľov

Aj tento server tak isto pracuje s xml databázou užívateľov, ktorá je uložená do súboru userlist.xml. Tento súbor má byť uložený v adresári *C:\Temp*. V tomto prípade trieda User obsahuje metódy pre prácu s databázou. Xml súbor userlist.xml je načítaný pomocou metódy LoadXml( ), ale v tomto prípade v triede XmlDocument bola už preddefinovaná metóda Load( ) na načítanie xml súboru. Pridávanie a vymazávanie užívateľov je podobne riešené ako v Jave, tu sa zavolajú metódy AddUser( ), resp. DeleteUser( ), ktoré upravujú obsah xml súboru.

Zoznam užívateľov je vypísaný do tzv. DataGridView, ktorý je podobný k tabuľke. Jeho obsah sa aktualizuje pomocou metódy GetUserList( ).

3.3.3 Spustenie a zastavenie servera

Na spustenie servera slúži tlačítko *Start server*:

```
ftpServer = new FtpServer();
FtpServerThread = new Thread(new ThreadStart(ftpServer.Connect));
FtpServerThread.Start();
```

Po stlačení tlačítka sa vytvorí nový konštruktor typu FtpServer a pustí sa jeho vlákno, to je metóda Connect( ).

```
1) serverSocket = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Tcp);
2) IPEndPoint serverIPEP = new IPEndPoint(IPAddress.Any, 21);
3) serverSocket.Bind(serverIPEP);
4) serverSocket.Listen(4);
```

V konštruktor ftpServer je najprv vytvorený jeden štandardný socket , čo v 3) priviažem ku konkrétnemu koncovému bodu, kde je už možné špecifikovať ip adresu a port (samozrejme je používaný port 21, lebo ide o FTP). V 4) príkazom Listen( ) odkážem socketu aby začal počúvať na danom porte a čaká na sockety klientov, tým pádom sa začne správať ako serversocket.

```
1) Socket controlSocket = serverSocket.Accept();
2) IPEndPoint controlSocketIPEP =
        (IPEndPoint)controlSocket.RemoteEndPoint;
3) clientIp = controlSocketIPEP.Address;
4) Connection Connection = new Connection(controlSocket, i, this,
         clientIp);
5) Thread connectionThread = new Thread(new
   ThreadStart(Connection.Run));
6) connectionThread.Start();
```

Aj v tomto prípade tak isto funguje vytváranie riadiaceho spojenia ako v Jave. V 1) serverSocket dovtedy čaká, kým sa pripojí k nemu nejaký klient, potom ho akceptuje a v 4) sa vytvorí konštruktor triedy Connection, ktorý reprezentuje riadiace spojenie. V 5) a 6) sa vytvorí a spustí vlákno riadiaceho spojenia, kde sú prenášané všetky príkazy pri komunikácii s klientom.

Na zastavenie servera slúži tlačítko *Stop server*:

```
serverSocket.Close();
```

Zavolaná metóda zatvorí serverov socket, a tým pádom je server zastavený.

3.3.4 Pripojovanie klienta (riadiace spojenie)

Keď je server už spustený a pripravený na prijímanie spojení, tak sa užívateľ môže pripojovať k danej ip adresy a k portu so svojím prihlasovacím menom a heslom. Server vo svojom vlákne najprv akceptuje klientov socket, a potom vytvorí nové riadiace spojenie s ním, čím sa pustí ďalšie vlákno pre toto spojenie s aktuálnym klientom.

```
NetworkStream inStream = new NetworkStream(controlSocket);
StreamReader controlIn = new StreamReader(inStream);
NetworkStream outStream = new NetworkStream(controlSocket);
controlOut = new StreamWriter(outStream);
```

V tomto prípade komunikácia cez riadiace spojenie je vyriešené pomocou špeciálnych streamov, tzv. networkstreamov, ktoré sú doporučené v jazyku C# pri komunikácii cez sieť. Na prijímanie týchto streamov je používaná premenná, typu StreamReader a na odoslanie streamov ku klientovi slúži contolOut, typu StreamWriter. Cez toto spojenie prechádzajú všetky príkazy FTP, ktoré sú potrebné pre pripojovanie klienta a pre rôzne iné služby okrem dátových.

3.3.5 Prenášanie dát (dátové spojenie)

(FilaZilla klient štandardne pracuje v pasívnom režime, takže predpokladajme, že ide o pasívny režim – aktívny režim by bol podobný.)

Server po prijatí príkazu PASV sa pripraví na pasívny režim:

```
dataListener = new DataListener(serverIp);
dataListenerThread = new Thread(new ThreadStart(dataListener.Run));
dataListenerThread.Start();
```

Vytvorí nový konštruktor triedy DataListener so serverovou ip adresou, v ktorom si vytvorí nový serverSocket pre prípadný dátový prenos. Vytvorí sa nové vlákno tohto konštruktoru, v ktorom server čaká na danom ip adrese a porte na prípadný prenos, a potom sa aj pustí.

```
dataConnection = new DataConnection(this, dataListener.GetSocket(),
         DataAction.RETR, fileDir);
dataConnectionThread = new Thread(new
         ThreadStart(dataConnection.Run));
dataConnectionThread.Start();
```

Prenášanie dát je realizované vo vlákne konštruktoru triedy DataConnection, kde sa uskutočnia oba typy prenosu (retrieve a store).

3.4 Sieťové programovanie

3.4.1 Platforma JavaSE

K vývoju programu na platforme JavaSE som hlavne používal balíky java.net a java.io.

Balík java.net obsahuje triedy k vytvoreniu sieťových komponentov na základe TCP. Sockety servera (trieda ServerSocket) a klienta (trieda Socket) môžeme vytvárať so špecifikovaním ich portu a ip adresy (trieda InetAddress).

K sieťovému programovaniu je ešte dôležitý balík java.io, v ktorom sú implementované vstupy a výstupy jednotlivých socketov. Sieťová komunikácia sa uskutočňuje pomocou streamov. Prichádzajúce streamy sú čítané zo vstupu pomocou triedy InputStreamReader, a hneď sú ukladané do medzipamäte (trieda BufferedReader). Trieda PrintWriter umožňuje vypísanie streamov na výstup a tým posielanie dát ku klientovi.

3.4.2 Platforma .NET

Na tejto platforme k sieťovému programovaniu sú dôležité namespacy System.Net, System.Net.Socket a System.IO.

Namespace System.Net.Socket obsahuje triedy Socket a NetworkStream. Prvá umožňuje vytvárať nové sockety, druhá reprezentuje špeciálne streamy, ktoré sú posielané v dátovom komunikácii cez sieť. Po vytvoreniu socketu musíme ho viazať ku konkrétnemu koncovému bodu (trieda IPEndPoint), ktorý je definovaný s ip adresou (trieda IPAddress) a číslom portu. V prípade socketu servera ešte mu máme odkázať aby začal počúvať na danom porte a čakať na prichádzajúce sockety od klienta. Posledné dve zmienené triedy sú implementované v namespace System.Net.

Posledný namespace obsahuje triedy na vypísanie streamov na výstup socketu (trieda StreamWriter) a na čítanie streamov zo vstupu socketu (StreamReader).

3.4.3 Porovnanie

Tieto dve platformy používajú podobné spôsoby a triedy k vývoju serverovej aplikácii, ale samozrejme sú medzi nimi menšie rozdiely.

Najvýraznejší rozdiel je vo vytváraniach socketov. V Jave vytvorenému socketu môžeme zadávať parametre ako ip adresa a číslo portu, kým v .NETe vytvorený socket musíme viazať ku konkrétnemu koncovému bodu, kde sa potom špecifikuje ip adresa a číslo portu. Ďalším rozdielom je, že Java obsahuje zvlášť triedu pre serversocket, ktorý po vytvoreniu automaticky začne počúvať na danom porte, kým v .NETe štandardnému socketu môžeme odkázať aby začal počúvať na danom porte, a tým pádom sa správal ako serversocket.

V komunikácii sieťových komponentov v .NETu sú presnejšie definované streamy pre sieťovú komunikáciu (tzv. networkstreamy) ako v Jave, kde sa používajú štandardné streamy (InputStream a OutputStream).

4 Základy sieťovania a vlastnosti FTP

4.1 Základy sieťovania

4.1.1 Komunikačné protokoly

Počítače, ktoré komunikujú medzi sebou na internete používajú aj Transmission Control Protocol (TCP) aj User Datagram Protocol (UDP).

Aplikačná vrstva (HTTP, FTP, Telnet, ...)
Transportná vrstva (TCP, UDP)
Sieťová vrstva (IP, ICMP)
Vrstva sieťového rozhrania (protokoly sieťového rozhrania)

Obr. 8: Vrstvy architektúry TCP/IP

Keď píšeme program, ktorý komunikuje cez sieť, tak programujme v aplikačnej vrstve, teda nemusíme sa dotknúť k TCP a UDP vrstvám. Namiesto toho môžeme používať triedy z daných balíkov alebo namespacov. Tieto triedy nám poskytujú systém nezávislú sieťovú komunikáciu.

TCP je zabezpečený protokol, ktorý poskytuje spoľahlivý dátový tok medzi dvomi počítačmi. Poskytuje point-to-point kanál pre aplikácie, ktoré vyžadujú spoľahlivú komunikáciu. Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP) a Telnet sú príkladmi na takýto prenos.

UDP je taký protokol, ktorý posiela nezávislé pakety, nazývané datagramy, z jedného počítača k druhému so žiadnou zárukou doručenia. Posielanie datagamov je niečo podobné ako posielať listy s poštou. Dodací list nie je garantovaný, že sa dostane k príjemcovi a každý list je nezávislý od druhého. Kvôli viacerým aplikáciám garantovanie spoľahlivosti je dôležité

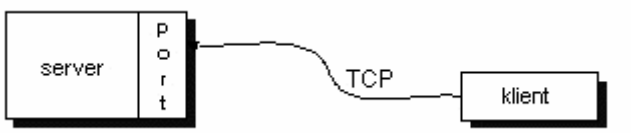
k úspešnému prenosu informácii, ale sú aj také aplikácie ktoré nepotrebujú také prísne štandardy ako napríklad hodinový server.

4.1.2 Porty

Všeobecne povedané, počítač má fyzické spojenie k sieti. Všetky dáta nasmerované k partikulárnemu počítaču idú cez toto spojenie. Hoci dáta sú určené k rôznym aplikáciám bežiacim na počítači, ale on to rozpozná pomocou portov.

Dáta prenášané cez Internet sú doložené adresovaním informácie, podľa čoho rozpozná počítač a port kam je nasmerovaný. Počítač je adresovaný s 32-bitovou IP adresou a porty 16-bitovým číslom.

V spoľahlivej komunikácii ako TCP, je serverová aplikácia naviazaná socketom pre špecifické číslo portu. Toto registruje server so systémom pre prijatie všetkých dát nasmerované k portu a klient vie komunikovať so serverom na tomto porte.



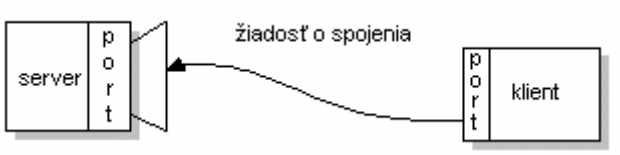
Obr. 9: Pripojený klient k portu serveru pomocou TCP

Porty sú označované od 0 do 65 535, lebo sú predstavované s 16-bitovým číslami. Niektoré čísla sú zarezervované a len systémové procesy ich môžu používať. Na začiatku boli vyhradené čísla 0 – 255, ale po určitej dobe sa museli rozšíriť na 0 – 1023. Tieto porty sú nazývané „Well known ports“ (mohli by sme preložiť ako „známe porty”).

4.1.3 Sockety

Bežne server beží na špecifickom počítači a má socket, ktorý je naviazaný na špecifické číslo portu. Server čaká na žiadosť o pripojenie od klienta cez tento socket.

Na strane klienta: Klient vie názov hostu na ktorom server beží a číslo portu na ktorom server čaká na klienta. Pre žiadosť o pripojenie, klient sa pokúsi stretnúť so serverom na serverovom porte. Klient tiež potrebuje identifikovať server a naviaže sa na lokálny port, ktorý bude ďalej používať, ktorý väčšinou prideľuje systém.



Obr. 11: Žiadosť klienta o spojenia k danému portu

Keď je všetko v poriadku, tak server prijme spojenie. Po prijatí server získa nový socket viazaný k tomu istému lokálnemu portu, a má tiež svoj vzdialený koncový bod nastavený na adresu a port klienta. Potrebuje nový socket, takže vie pokračovať v počúvaní v pôvodnom sockete na žiadosť počas zásobovania pripojeného klienta.

Obr. 12: Naviazané spojenie so serverom

Keď je spojenie prijaté na strane klienta, tak socket je úspešne vytvorený a klient môže používať tento socket na komunikáciu so serverom. Takže teraz už klient vie komunikovať s písaním a čítaním zo svojho socketu.

Koncový bod je kombináciou jednej IP adresy a čísla portu. Každé TCP spojenie môže byť jedinečne identifikované so svojimi koncovými bodmi. S týmto spôsobom môžeme mať viacnásobné spojenia medzi hostom a serverom.

4.2 File Transfer Protocol (FTP)

4.2.1 Obecná charakteristika FTP

FTP (File Transfer Protocol) je dobre stanovený internetový protokol určený na prenášanie dáta (a informácie o súboroch) cez počítačové siete, ktoré používajú TCP.

Prvá definícia FTP bola vytvorená v roku 1971 ako súčasť projektu ARPANET. Vývoj prebiehal štrnásť rokov až do roku 1985, kedy bol publikovaný posledný rfc dokument zaoberajúci sa základnou špecifikáciou FTP. Od tej doby sa pokračuje na vývoji

najrôznejšieho rozšírenia, ale základná definícia protokolu je už takmer dvadsať rokov nezmenená. Tento dokument je Request For Comments 959 (RFC 959), ktorý je pre každého dostupný z internetovej stránky Internet Engineering Task Force (www.ietf.org).

Pri vývoji FTP protokolu stáli nasledovné ciele: zaistiť zdieľanie súborov, nepriame ovládanie vzdialených počítačov, chrániť užívateľov pred zvláštnosťami a rozdielmi súborových systémov na vzdialených počítačoch a prenášať efektívne a bezpečne dáta. Dnes sa FTP používa prevažne k zdieľaniu a prenosu súborov.

FTP potrebuje jednu klientovú (FTP client) a jednu serverovú aplikáciu (FTP server). Klient môže načítať súbory a informácie o súboroch zo serveru a tiež nahrávať súbory na server, ktorý je väčšinou chránený heslom.

4.2.2 FTP operačný model, komponenty protokolov a kľúčové terminológie

Štandardy, ktoré sú definované FTP, popisujú jeho obecnú operáciu s použitím jednoduchého nástroja, tzv. FTP model. Tento model definuje role jednotlivých komponentov, ktoré sa zúčastňujú v dátovom prenose a dve komunikačné spojenia, ktoré sú vytvorené medzi nimi. Tiež popisuje komponentov, ktoré riadia tieto spojenia a definuje používané terminológie komponentov.

FTP-SERVER PROCES A FTP-USER PROCES

FTP je klasický klient/server protokol, kde klient je nazývaný ako user (užívateľ). Názov vychádza z toho, že užívateľ zadáva ftp príkazy na počítači klienta. Celá sada používaných FTP softwareov, sa nazýva proces. FTP software na serveri sa nazýva Server-FTP process (FTP-server proces) a to isté User-FTP Process (FTP-user proces) na strane klienta.

FTP RIADIACE SPOJENIE A DÁTOVÉ SPOJENIE

FTP ako veľa iných aplikácii používa TCP, ale nepoužíva len jedno TCP spojenie pre všetky komunikačné cesty ako väčšina protokolov. FTP model je skonštruovaný okolo dvoch spojeniach komunikácií medzi procesmi servera a klienta:

Riadiace spojenie (control connection): Toto hlavné logické spojenie je vytvorené pri začatí FTP spojenia. Je udržované behom celého FTP spojenia a je používané na prechod

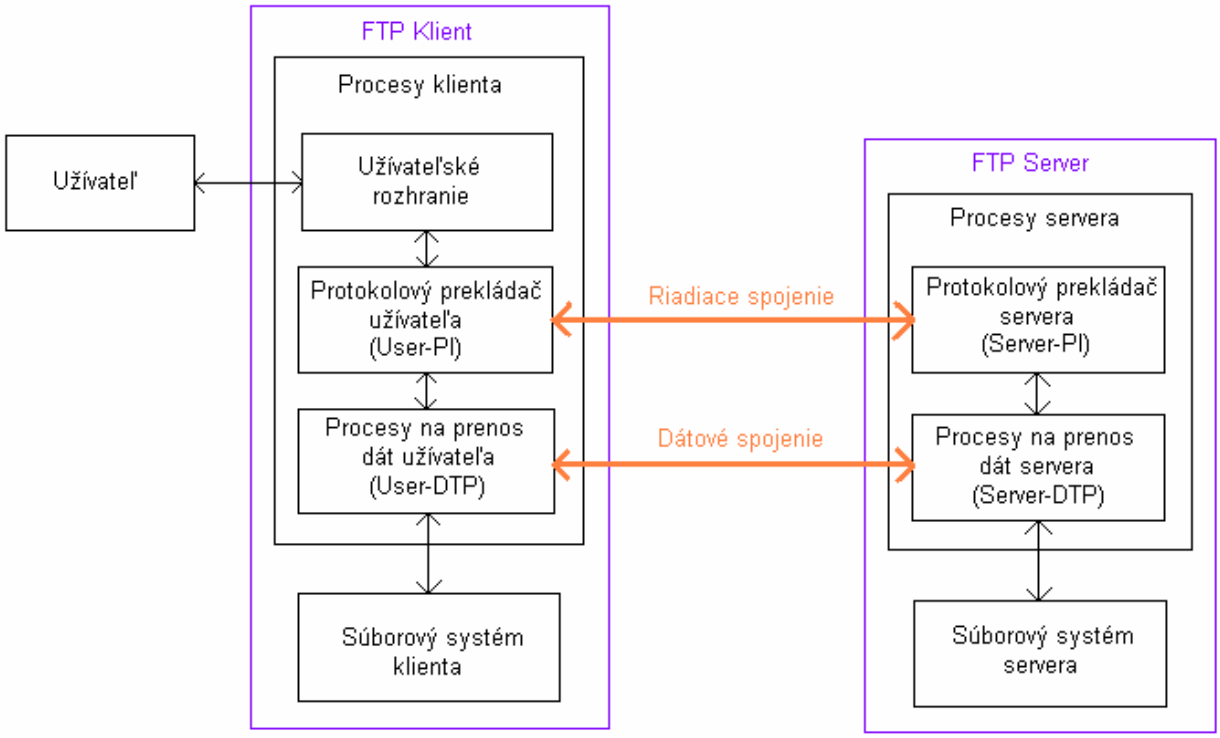
kontrolných informácii, ako FTP príkazy a odpovede. Nie je používané na prenášanie súborov.

Dátové spojenie (data connection): Vždy, keď sú dáta poslané od servera ku klientovi, alebo opačne, TCP dátové spojenie bude vytvorené medzi nimi a dáta sú prenášané cez toto spojenie. Keď je prenos dokončený, tak sa spojenie ukončí.

FTP PROCESOVÉ KOMPONENTY A TERMINOLÓGIE

Odvtedy ako riadiace a dátové funkcie používajú odlišné spojenia, software je delený do dvoch logických protokolových komponentov na obidvoch stranách, ktoré sú zodpovedné za jednotlivé spojenia. Protocol interpreter - protokolový prekladač (PI) - je software, ktorý ovláda riadiaceho spojenia, posiela a prijíma príkazy a odpovede. Data transfer process - proces prenosu (DTP) - je zodpovedný za aktuálne posielanie a prijímanie dáta medzi klientom a serverom. Dodatkom k týmto dvom prvkom, je aj tretí komponent, user interface (užívateľské rozhranie).

Teda, v FTP sú dva komponenty procesu servera a tri komponenty procesu klienta. Tieto sú uvedené v modeli FTP so špecifickými názvami, ktoré sú v štandarde používané na popísanie detailných operácii protokolu. Jednotlivé komponenty sú podrobnejšie vysvetlené na nasledujúcom obrázku.



Obr. 13: Operačný model FTP

FTP je protokol typu klient/server s komunikáciou medzi User-FTP Process na strane klienta a Server-FTP Process na strane servera. Príkazy, odpovede a stavové informácie sú posielané cez vytvorené riadiace spojenie (control connection) medzi User-PI a Server-PI. Dáta sú prenášané medzi dvoma stranami cez dátové spojenie (data connection), ktoré je vždy pre každý nový prenos znova vytvorené.

KOMPONENTY PROCESU SERVER-FTP

Tento proces obsahuje nasledujúce dva protokolové prvky:

Server protocol interpreter - Serverový protokolový prekladač (Server-PI): Prekladač protokolu je zodpovedný za ovládanie riadiaceho spojenia na strane servera. Na rezervovanom porte čaká na prichádzajúce spojenie od klientov. Keď bolo spojenie úspešne vytvorené, tak potom prijíma príkazy od User-PI, posiela naspäť odpovede a riadi dátové prenosy servera.

Server data transfer process - Serverov proces prenosu (Server-DTP): DTP na strane servera je používané na posielanie a prijímanie dáta smerujúce od alebo k User-DTP. Server-DTP. Môže tiež vytvoriť dátové spojenie, alebo čakať na dátové spojenie prichádzajúceho od klienta.

KOMPONENTY PROCESU USER-FTP

Tento proces obsahuje tieto tri protokoly:

User protocol interpreter - Klientov protokolový prekladač (User-PI): Tento protokolový prekladač je zodpovedný za riadenie riadiaceho spojenia na strane klienta. Toto inicializuje FTP spojenia s poslaním žiadosti k Server-PI. Po vytvorení spojenia spracováva prijaté príkazy od rozhrania klienta, posiela ich k Server-PI, a potom prijíma odpovede. Tiež riadi prenosový proces klienta.

User data transfer process - Prenosový proces klienta (User-DTP): DTP na strane klienta, ktoré posiela a prijíma dáta smerujúce od alebo k Server-DTP. User-DTP môže tiež vytvoriť dátové spojenie alebo čakať na dátové spojenie prichádzajúce od servera.

User interface – Užívateľské rozhranie: Poskytuje jedno „prijateľnejšie“ FTP rozhranie. Dovoľuje jednoduchšie pochopenie príkazov pre užívateľa používané pre funkciu FTP ako zložité FTP príkazy a tiež povoľuje výsledky a informácie sprostredkovať späť na druhú stranu spojenia.

4.2.3 FTP príkazy

Pre pochopenie priebehu spojenia a prenosu súborov treba najprv vysvetliť FTP príkazy. Tieto príkazy sú posielané cez riadiace spojenie z jednej strany spojenia na druhú alebo opačne, skladajú sa z jednoduchých textových reťazcov. Obsahujú štyri znaky a klient nimi spustí a riadi spoluprácu so serverom.

Prijaté príkazy sú vždy potvrdené odpoveďou FTP servera, ktorá sa skladá z trojciferného čísla nasledovaným s nejakým čitateľným textom. Prvá cifra ukazuje či je odpoveď dobrá, zlá alebo nedokončená. Keď nastane chyba, tak druhá a tretia cifra je používaná na ukazovanie typu a spresnenie chyby.

V nasledujúcich odstavoch sú krátko vysvetlené významy tých najdôležitejších príkazov pri komunikácii. Ostatné príkazy sú podrobnejšie popísané v referencii (RFC959).

USER NAME (*USER*) – užívateľské meno

Použitie: *'USER užívateľské_meno'*

Pred zahájením práce s FTP serverom je potrebné prihlásiť sa týmto príkazom. Príkaz *USER* možno použiť kedykoľvek pri práci so serverom a ihneď tak zmeniť aktívneho užívateľa.

PASSWORD (*PASS*) – heslo

Použitie: *'PASS heslo'*

Tento príkaz musí nasledovať po príkaze *USER*. Užívateľské rozhranie klienta by malo byť chránené heslom, a ešte heslo by malo byť zašifrované.

LOGOUT (*QUIT*) – odhlásenie

Použitie: *'QUIT'*

Tento príkaz ukončí prácu užívateľa a pokiaľ sa neprenáša žiadny súbor ukončí riadiace spojenie. Pokiaľ je súbor prenášaný, server po potrebnej dobe ponechá spojenie pre hlásenie výsledkov prenosu.

PASSIVE (*PASV*) – pasívne spojenie

Použitie: *´PASV´*

Klient informuje servera, že sa chce pripojovať pasívne. Žiada od servera správu, že k akej ip adrese a ku ktorému portu sa má klient pripojovať.

DATA PORT (*PORT*) – port dátového spojenia

Použitie: *'PORT h1,h2,h3,h4,p1,p2'*

Tento príkaz je používaný len v aktívnom režime. Špecifikuje adresu a port, že kde bude čakať klient na dátové spojenie zostavované serverom. Čísla h1,h2,h3,h4 špecifikujú IP adresu klienta, čísla p1,p2 port, na ktorý sa má server pripojovať. Číslo portu sa určí pomocou funkcie, kedy číslo p1 v binárnom tvaru sa posunie doľava o osem miesto, a tieto miesta sa vyplnia nulou. Potom namiesto týchto núl zapíšeme číslo p2 v binárnom tvaru. To znamená,

že číslo p1 je hodnota prvých ôsmych bitov zo šestnásť bitového čísla portu a číslo p2 druhých osem bitov.

Túto funkciu a vypočítanie portov ukážem na nasledujúcom príklade:
Zvolím si náhodne dve čísla p1 = 53 a p2 = 194.
prevediem p1 do binárneho tvaru: p1_binárne = 01000101
tak isto p2 v binárnom tvare: p2_binárne = 10000010
posuniem p1_binárne o 8 miest doľava, takže jeho miesto doplním samými nulami:
p1_binárne_posunuté = 0100010100000000
potom pričítam binárne číslo p2_binárne, z čoho už dostanem číslo portu:
port_binárne = p1_binárne_posunuté + p2_binárne = 0100010110000010
na konci treba ešte toto číslo previesť do dekadického tvaru:
port = 13346.

TYPE (*TYPE*) – typ prenášaného súboru
Použitie: *'TYPE typ_charakteru'*

Nastaví typ prenášaného súboru. So zadaním príkazov *binary* a *ascii* môžeme nastavovať typ prenášaných súborov na binárny alebo ASCII. Po nastavení typu program vypíše pomocou príkazu *out.println("200 type set")* s nami zvolený typ.

PRINT WORKING DIRECTORY (*PWD*)
Použitie: *´PWD názov_adresára´*

Vypíše sa názov aktuálneho pracovného adresára na serveri, ale jeho obsah nie.

CHANGE WORKING DIRECTORY (*CWD*) – zmena pracovného adresára
Použitie: *'CWD názov_adresára'*

Pomocou tohto príkazu sa zmení aktuálny adresár na adresár zadaný príkazom. V správe je napísaný názov adresára.

CHANGE DIRECTORY UP (*CDUP*)
Použitie: *'CWD názov_adresára'*

Zmena na nadriadený adresár.

DELETE (*DELE*) – vymazať súbor
Použitie: *'DELE názov_súboru'*

Príkaz *DELE* vymaže súbor na serveri. Užívateľ si vyberie súbor na vymazanie, a potom príkazom *delete()* je inštrukcia prevedená.

RETRIEVE (*RETR*) – získať súbor
Použitie: *'RETR názov_súboru'*

Tento príkaz spôsobí, že Server-DTP prenesie súbor uvedený v príkaze na server alebo User-DTP na druhom konci dátového spojenia. Súbor na serveri sa nezmení.

STORE (*STOR*) – uložiť súbor
Použitie: *'STOR názov_souboru'*

Tento príkaz spôsobí, že server prijme dáta prenášané cez dátové spojenia a uloží tieto dáta ako súbor v aktuálnom adresári serveru. Pokiaľ na serveri už takýto súbor existuje, bude nahradený novým prijatím.

LIST (*LIST*) – vypísanie obsahu aktuálneho adresára
Použitie: *'LIST názov_adresára'*

Metóda *LIST* získa zoznam súborov (a možných adresárov). Keď adresár je špecifikovaný, tak zoznam súborov v danom adresári je poslaný serverom naspäť spolu so špecifikovanou informáciou o súboroch. Poslaná informácia bude záležať na systéme servera. Tento zoznam by mal byť v dátovom type ASCII. Keď adresár nie je špecifikovaný, tak sú poslané aj podrobnosti o aktuálnom pracovnom adresári.

4.2.4 Riadiace spojenie a autentizácia klienta

FTP operačný model popisuje logické dáta a riadiace spojenie, ktoré je vytvárané medzi klientom a serverom. Predtým ako je možné používať dátové spojenie na prenášanie súborov, musí byť vytvorené riadiace spojenie. Špecifický proces je nasledovaný založením tohto spojenia a tým je vytvorené permanentné FTP spojenie medzi dvoma stranami, cez ktoré už môže byť uskutočnený prenos.

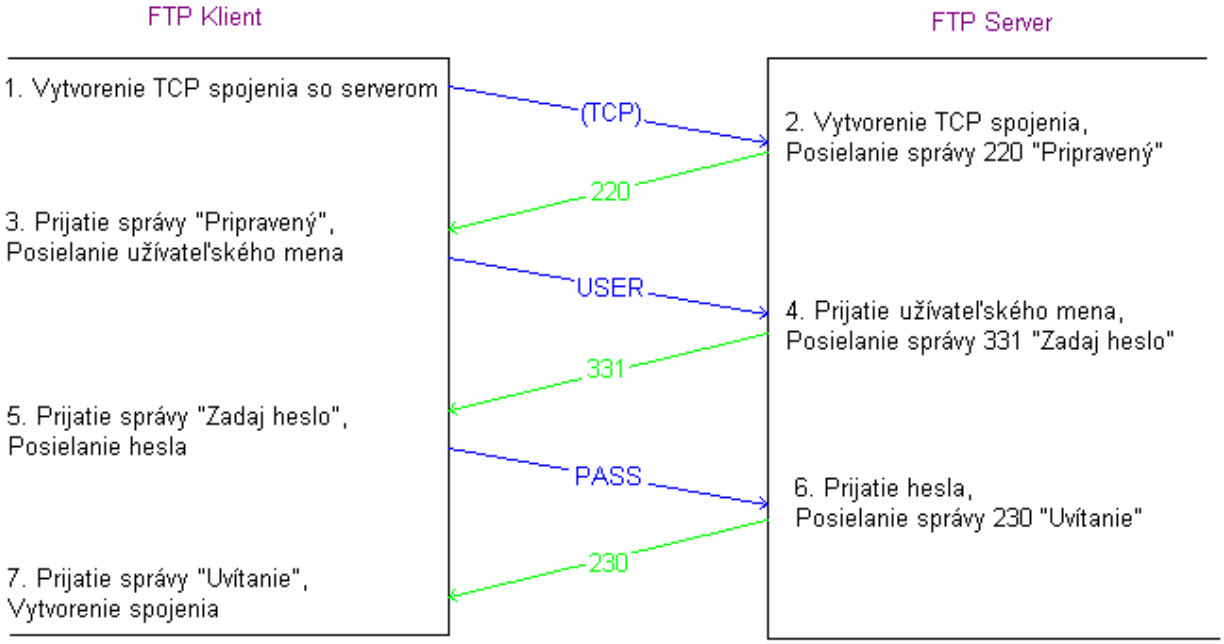
Tak ako v ostatných klient/server protokoloch FTP server má pasívnu rolu v procesoch riadiaceho spojenia. Server-PI počúva na špeciálnom porte, rezervovaný pre FTP riadiace spojenie: port 21. User-PI inicializuje spojenie s otvorením TCP spojenia na tomto porte od užívateľa k serveru.

Keď je už TCP raz založené, tak riadiace spojenie je vytvorené medzi dvoma stranami, takže príkazy a odpovede môžu byť posielané medzi User-PI a Server-PI. Prvá vec po vytvorenia spojenia je autentizácia užívateľa, ktoré je nazývané v FTP ako *login sequence*. Existujú dve možnosti autentizácii:

Access Control - Kontrola prihlasovania: Autentizačný proces dovoľuje sa prihlasovať k serveru len pre autorizovaných užívateľov a server vie kontrolovať užívateľov.

Resource Selection - Výber zdroja: S identifikovaním užívateľov FTP server sa môže rozhodovať, ku ktorým zdrojom pustí užívateľa.

Regulárna autentizačná schéma FTP je jednoduchá "username/password" prihlasovacia schéma, znázornená na obrázku. S touto metódou sa môžeme stretávať na mnohých internetových stránkach. Užívateľ je najprv identifikovaný s príkazom *USER* od User-PI k Server-IP, to je poslanie užívateľského mena. Potom je heslo poslané pomocou príkazu *PASS*.



Obr. 14: Vytvorenie FTP spojenia a užívateľská autentizácia

Začína sa s vytvorením spojenia TCP medzi klientom a serverom (Vytvorenie TCP spojenia so serverom). Klient potom pošle meno užívateľa a heslo (Posielanie užívateľského mena, Posielanie hesla), a server ich overí. Keď informácie sú prijaté so serverom (Prijatie užívateľského mena, Prijatie hesla), ten odpovedá správou že je všetko v poriadku a spojenie sa môže otvoriť (Posielanie správy 230 “Uvítanie“).

Server vyhľadá užívateľské meno a heslo v databáze, aby overil či prihlasujúci užívateľ má platné právo pripojiť sa na server. Keď je informácia platná, tak server signalizuje otvorenie spojenia s kladnou odpoveďou. Keď sa klient prihlasuje s nesprávnym užívateľským menom a heslom, tak ho server nepustí ďalej. Po určitom počte zadaných nesprávnych údajov, server môže odmietnuť ďalšie pokusy prihlasovania.

Každý užívateľ môže mať, závisiace od nastavenie servera, svoj vlastný vyhradený priestor na serveru alebo práva obmedzené len na dátové prenosy. Servery umožňujú špecifikovať rôzne vlastnosti užívateľských účtov. Možné je každému účtu nastaviť rôzne vlastnosti, napr. maximálnu dovolenú rýchlosť sťahovania dát alebo maximálnu veľkosť diskového priestoru, ktorý je užívateľovi k dispozícii, atď.

Keď prihlasovanie prebehlo úspešne, tak server vytvorí spojenie vzhľadom nato, že aké práva klient dostal pri prihlasovaní. Niektorí užívatelia môžu dostať prístup len k pár súborom. Administrátor môže obmedziť prístup k FTP tak ako je potrebné.

Keď je už spojenie vytvorené, tak si server môže vyberať na základe identity užívateľov. Napríklad, na systéme s viacerými užívateľmi, administrátor môže vytvoriť FTP vtedy, keď sa nejaký užívateľ prihlási a on sa automaticky dostane do svojho domovského adresára.

4.2.5 Dátové spojenie a režimy prenosov

Riadiace spojenie vytvorené medzi Server-PI a User-PI používa proces na založenie spojenia FTP a na autentizáciu. Príkazy a odpovede sú vymieňané medzi protokolovými prekladačmi cez toto spojenie, ale dáta potrebujú nové spojenie.

Dátové spojenie musí byť vždy vytvorené, keď súbory alebo iná dáta sú prenášané medzi serverom a klientom. Toto spojenie je potrebné pri prenášaniach súborov, implicitných prenášaniach dát, aj pri požiadaní vypísanie súborov adresára zo serveru.

Štandard FTP špecifikuje dve odlišné cesty vytvorenia dátového spojenia. Tieto dve metódy sa odlišujú v tom, že na ktorej strane je spojenie (na serverovej alebo na klientskej) zahájené. Režimy sa vzťahujú na činnosť FTP servera, a nie klienta.

AKTÍVNE DÁTOVÉ SPOJENIE

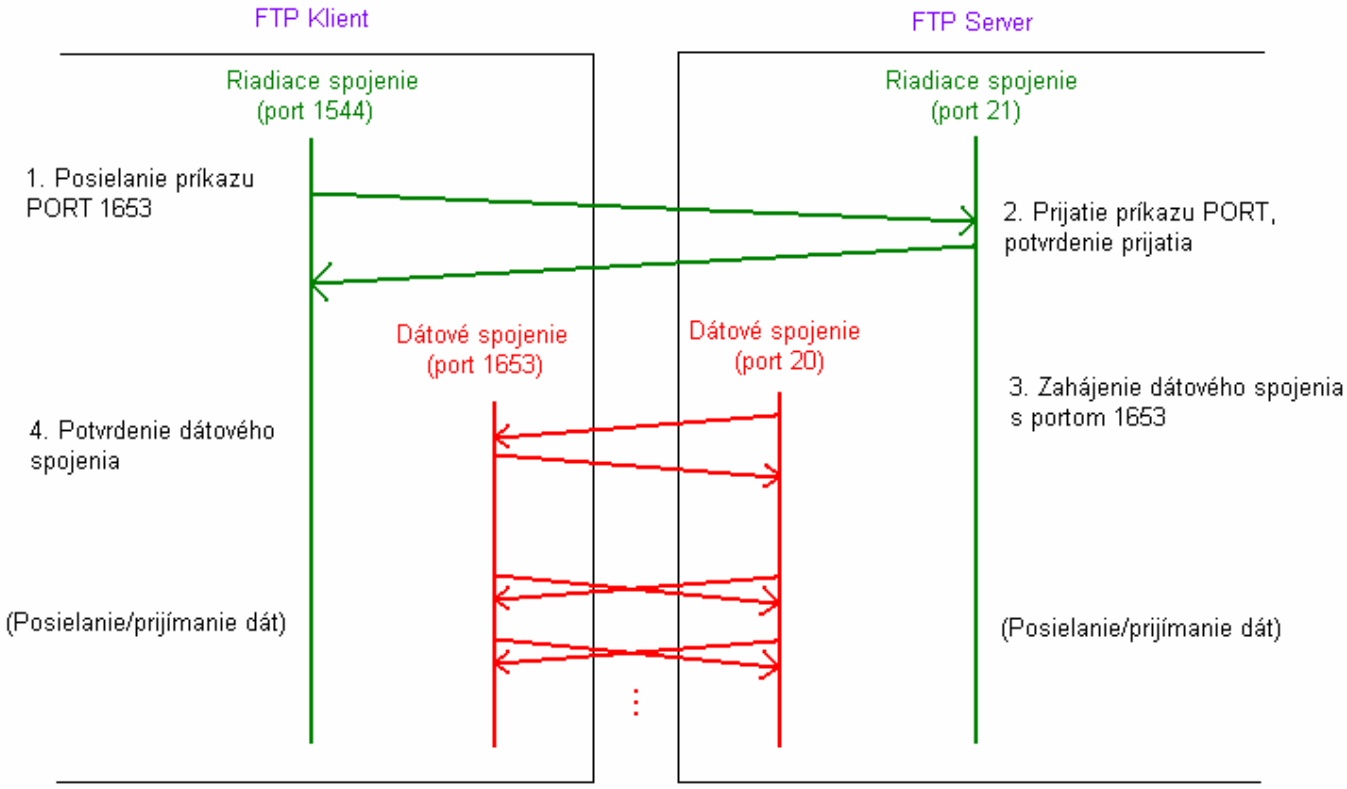
Prvá metóda môže byť nazývaná ako vytvorenie normálneho dátového spojenia (lebo táto je štandardnou metódou), alebo ako aktívne dátové spojenie (v porovnaním s pasívnou metódou).

V tomto type spojenia, Server-DTP zahájí dátový kanál s otvorením TCP spojenia pre User-DTP. Server používa špeciálne rezervovaný port 20 pre dátové spojenia. Na strane klienta štandardné číslo portu je to isté ako dočasne používaný port pre riadiace spojenie, ale klient si často na každý prenos vyberie nejaký iný port.

Situáciu znázorňujem na nasledujúcom príklade. Predpokladane User-PI vytvoril riadiace spojenie z portu 1544 na serverový FTP kontrolný port 21. Ďalej na vytvorenie dátového spojenia pre dátový prenos, Server-PI dá pokyn pre Server-DTP na vytvorenie TCP

spojenia zo serverového portu 20 k portu klienta 1544. Po potvrdení klientom dáta môžu byť prenášané (v obidvoch smeroch, lebo TCP je dvojsmerný).

Prakticky nie je dobrý nápad, aby bolo riadiace aj dátové spojenie na rovnakom porte na strane klienta, lebo to komplikuje operáciu FTP a môže nastať problém. Z tohto dôvodu je doporučené, aby klient špecifikoval iné číslo portu používajúci príkaz *PORT* pred dátovým transferom. Napríklad, klient špecifikuje port 1653 používajúci *PORT*. Potom Server-DTP vytvorí spojenie z jeho portu 20 k portu 1653 klienta namiesto 1544. Toto je znázornené na nasledujúcom obrázku.



Obr. 15: Aktívne dátové spojenie

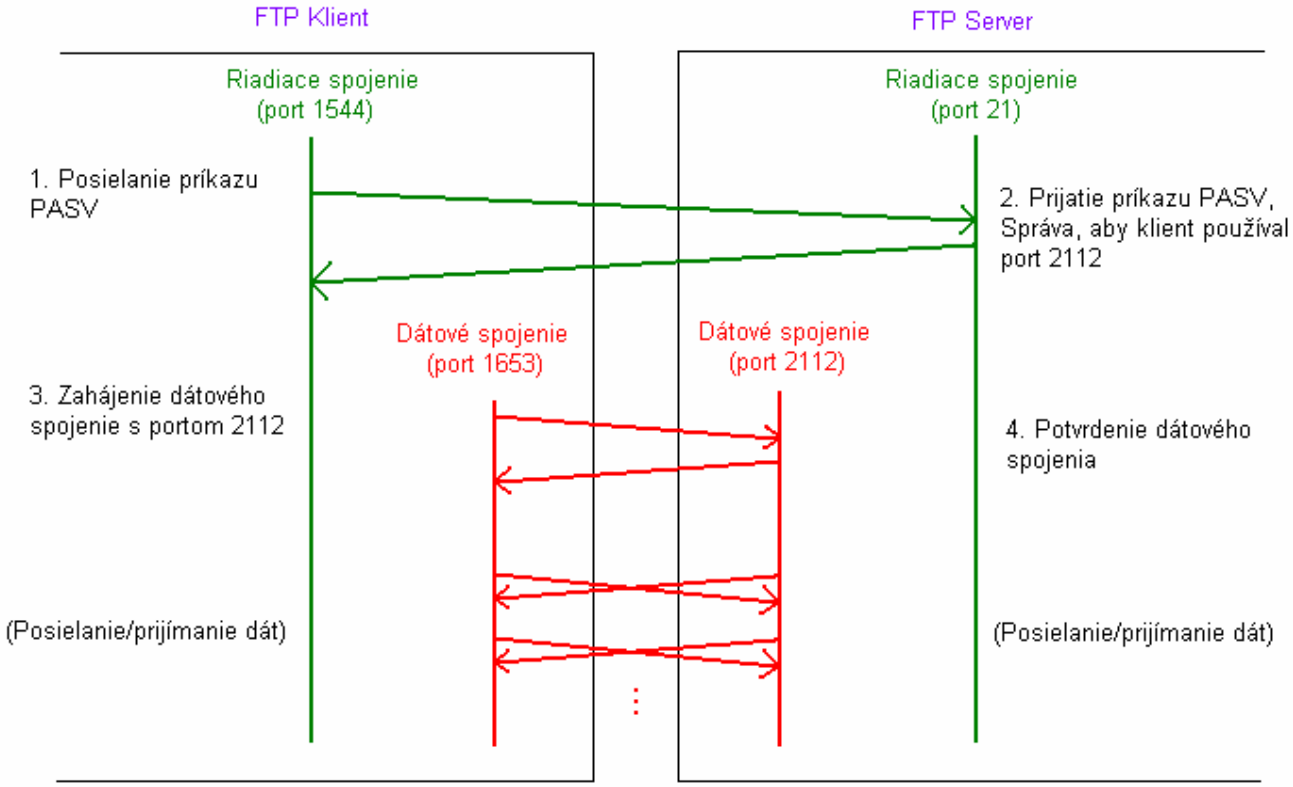
V aktívnom dátovom spojení server zahájí prenos dát s otvorením dátového spojenia klienta. V tomto prípade klient najprv pošle príkaz *PORT* (Posielanie príkazu *PORT* 1653), aby server vedel, že má používať port 1653 (Prijatie príkazu PORT, potvrdenie prijatia). Server potom otvorí dátové spojenie z jeho štandardného portu 20 k portu klienta 1653

(Zahájenie dátového spojenia s portom 1653). Dáta sú potom vymieňané na týchto dvoch portoch medzi dvoma stranami (Posielanie/prijatie dát).

PASÍVNE DÁTOVÉ SPOJENIE

Druhá metóda sa nazýva pasívne dátové spojenie. Server pasívne čaká na prijatie prichádzajúceho dátového spojenia inicializovaného klientom. Po úspešnom prijatí server odpovedá IP adresou a číslom portu, ktoré by mali byť používané. Server-DTP potom čaká na tomto porte na prichádzajúce TCP spojenie od User-DTP. Štandardne užívateľ používa ten istý port, ktorý bol použitý na riadiace spojenie, rovnako ako v aktívnom prípade. Ale keď je to nutné, tak klient si môže vybrať iný port na dátové spojenie.

Túto metódu znázorňujem na konkrétnom príklade. Klient pošle príkaz *PASV* (Posielanie príkazu *PASV*), aby dal najavo serveru, že chce používať pasívnu kontrolu dát. Server-PI odpovedá s tým, že klient má používať napríklad port 2112 (Správa, aby klient používal port 2112). Server-PI potom informuje Server-DTP, aby začal počúvať na porte 2112. User-PI informuje User-DTP o vytvorení spojenia z klientov portu 1653 k serverovému portu 2112 (Zahájenie dátového spojenia s portom 2112). Po potvrdení serverom (Potvrdenie dátového spojenia) dáta môžu byť poslané a prijaté zas v oboch smeroch (Posielanie/prijatie dát). Situácia je graficky znázornená na nasledujúcom obrázku.



Obr. 16: Pasívne dátové spojenie

4.2.6 Dátová komunikácia a prenosové režimy

Keď je už dátové spojenie vytvorené medzi komponentmi Server-DTP a User-DTP, dáta sú prenášané priamo od klienta k serveru alebo opačne, závisí od špecifického príkazu. Od vyslania kontrolnej informácie cez riadiaceho spojenia, vytvorené dátové spojenie môže byť použité na dátovú komunikáciu.

FTP definuje tri rôzne prenosové režimy (transmission modes) čo presne špecifikuje, že ako sú dáta poslané z jednej strany na druhú cez otvorené dátové spojenie: *stream mode* (režim toku), *block mode* (blokový režim), *compressed mode* (komprimovaný režim).

STREAM MODE

V tomto režime dáta sú poslané ako plynulý stream bezštrukturovaných bajtov. Vysielacia strana proste začne tlačiť dáta cez TCP dátové spojenie k príjemcovi. Nie je používaný formát správ s rôznymi hlavičkami a preto je táto metóda trošku iná, lebo v iných protokoloch informácie sú poslané v diskrétnych dávkach. Toto závisí na dátovom streamingu a na spoľahlivých transportných službách TCP. Keď nie je používané záhlavie, tak koniec súboru je označený s tým, že po vykonaní práce vysielacie rozhranie ukončí dátové spojenie.

Z troch režimov je tento najčastejšie používaný v skutočných FTP realizáciách. Je štandardnou a najjednoduchšou metódou, takže je najjednoduchšie ju implementovať. Je najobecnejšia, lebo súbory upraví ako jednoduchý tok bajtov bez ohľadu na jeho obsah. Pracuje s najväčšou účinnosťou, lebo nie sú použité žiadne bajty na záhlavie.

BLOCK MODE

Dáta sú rozdelené do blokov a zabalené do jednotlivých FTP blokov. Každý blok má trojbajtové záhlavie, ktoré ukazuje jeho dĺžku a obsahuje ešte ďalšie informácie o bloku. Pomocou špeciálneho algoritmu sú sledované prenášané dáta, prípadne je zistený a reštartovaný prerušený prenos.

COMPRESSED MODE

Prenosový režim, v ktorom je používaná jedna relatívne jednoduchá komprimovacia technika tzv. *run-lenght encoding*. Zistí po sebe nasledujúce rovnaké bajty v prenášaných dátach, a potom ich kóduje skrátene, tým pádom výsledná správa zaberie menej bajtov. Skomprimovaná informácia je poslaná podobným spôsobom ako v blokovom režime.

Teoreticky sa zdá efektívny, ale praktický až tak veľmi nie, lebo komprimácia je často implementovaná v iných miestach siete, takže bude zbytočné v FTP. Napríklad, keď súbor je prenášaný cez internet s analógovým modemom, on bežne vykonáva kompresiu na prvej vrstve. Väčšie súbory na FTP serveroch sú tiež väčšinou komprimované, takže ďalšia kompresia by už nemala zmysel.

Záver

Mojou úlohou bolo preskúmať možnosti vývoja serverovej strany sieťových aplikácii na platformách JavaSE a .NET a vytvoriť na oboch platformách aplikáciu, ktorá zaistí základné služby FTP servera.

V prvej časti mojej práce som si podrobne preštudoval platformu JavaSE a možnosti sieťového programovania, ktoré sú popísané v balíkoch java.net a java.io. Po preštudovaní tejto platformy pomocou užívateľského prostredia NetBeans som naprogramoval FTP server.

V druhej časti som si preštudoval sieťové programovanie na platforme .NET, a zistil som veľa podobnosti s platformou JavaSE. Aplikáciu na tejto platforme som mohol vyvíjať podľa hotovej aplikácii v Jave. Mohol som používať podobné triedy, a zároveň sa aj syntax jazyka C# podobal na syntax jazyka Javy. Na programovanie tejto aplikácii som používal vývojové prostredie Visual Studio.

Po naprogramovaní oboch serverových aplikácií som zistil, že možnosti vývoja serverovej strany sieťových aplikácii oboch platforiem sú približne rovnaké, až na menšie detaily. Kým Java obsahuje zvlášť triedu pre vytváranie serverových socketov, v .NETe štandardnému socketu môžeme odkázať aby sa správal ako serversocket. Ďalší rozdiel je pri vytváraní socketov. V Jave môžeme priamo zadávať socketu parametre ako ip adresu a číslo portu, v .NETe socket viažeme ku koncovému bodu, a tam sa zadávajú tieto parametre. Na komunikáciu medzi jednotlivými sieťovými komponentmi sa používajú streamy. .NET má špeciálnu triedu na streamy, ktoré sú prenášané cez sieť, Java používa štandardné streamy.

Zistené rozdiely podľa mňa nie sú až tak výrazné, aby som mohol hovoriť o výhodách alebo nevýhodách jednej či druhej platformy.

Počas riešenia tohto projektu som sa naučil nielen možnosti vývoja serverovej strany na platforme JavaSE a .NET, priebeh a funkciu FTP, ale aj samostatne pracovať na rozsiahlejšom projekte.

Vedomosti, ktoré som získal počas riešenia bakalárskej práce, určite využijem v blízkej budúcnosti. Počas spracovávania tohto projektu som získal veľa skúseností s programovaním sieťových aplikácií na platformách JavaSE a .NET, ktoré ma zaujali a práve preto by som chcel svoje vedomosti ešte viac rozšíriť a využiť ich v rámci svojej praxe.

Literatúra

[1] HEROUT, P.: Učebnice jazyka JAVA, Kopp, České Budějovice 2000, 352s. ISBN 80-7232-115-3..

[2] HAROLD, E.R.: Java Network Progamming, O'Reilly, 2004, 760s. ISBN 0-596-00721-3.

[3] POSTEL, J., REYNOLDS, J., RFC 959 – File Transfer Protocol [online], The Internet Engineering Task Force, 1985. [cit. 2007-05-10]. Dostupné z URL: <http://www.ietf.org/ >.

[4] *TCPIPGUIDE.COM* [online]. c2007. [cit. 2007-05-10]. Dostupné z URL: <http://www.tcpipguide.com/>.

[5] VILNIUS, M.: C# pro zelenáče, Neocortex, 2002, 255s. ISBN 80-86330-11-7.